International Journal of Production Research

ISSN: 0020-7543 (Print) 1366-588X (Online) Journal homepage: https://www.tandfonline.com/loi/tprs20

An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system

Jun-qing Li, Shun-Chang Bai, Pei-yong Duan, Hong-yan Sang, Yu-yan Han & Zhi-xin Zheng

To cite this article: Jun-qing Li, Shun-Chang Bai, Pei-yong Duan, Hong-yan Sang, Yu-yan Han & Zhi-xin Zheng (2019): An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system, International Journal of Production Research, DOI: 10.1080/00207543.2019.1571687

To link to this article: https://doi.org/10.1080/00207543.2019.1571687

Published online: 12 Feb 2019.

International Journal of Production Research, 2019
https://doi.org/10.1080/00207543.2019.1571687

An improved artificial bee colony algorithm for addressing distributed flow shop with distance coefficient in a prefabricated system

Jun-qing Li[a,b*], Shun-Chang Bai[c], Pei-yong Duan[b], Hong-yan Sang[a], Yu-yan Han[a] and Zhi-xin Zheng[a]

[a]*School of Computer Science, Liaocheng University, Liaocheng, People's Republic of China;* [b]*School of Information and Engineering, Shandong Normal University, Jinan, People's Republic of China;* [c]*Department of Computing and Information Science, Cornell University, Ithaca, NY, USA*

(*Received 1 June 2018; accepted 25 December 2018*)

This paper proposes an improved artificial bee colony (IABC) algorithm for addressing the distributed flow shop considering the distance coefficient found in precast concrete production system, with the minimisation of the makespan. In the proposed algorithm, each solution is first represented by a two-dimensional vector, where the first dimensional vector is the factory and the second dimensional vector lists the operation scheduling sequence of each factory. Second, considering the distributed problem feature, a distributed iterated greedy heuristic (DIG) is developed where destruction and construction processes are designed in detail while considering the distributed structures. Third, an efficient population initialisation method that considers the factory workload balance is presented. Then, a local search approach that randomly replaces two factories with two randomly selected jobs and that finds an optimal position for the two inserted operations via the DIG method is proposed. For the canonical ABC algorithm, using the DIG approach, the main three parts are improved, namely, the employee, onlooker, and scout bees. Finally, the proposed algorithm is tested on sets of extended instances based on the well-known benchmarks. Through an analysis of the experimental results, the highly effective proposed IABC algorithm is compared to several efficient algorithms drawn from the literature.

Keywords: prefabricated production; flow shop; distributed scheduling; artificial bee colony algorithm; distance coefficient

1. Introduction

During recent years, many research works have focused on the green intelligent building optimisation problems, such as multi-zone HVAC system (Zeng, Zhang, and Kusiak 2015), optimal chiller loading optimisation problem (Duan et al. 2018; Zheng and Li 2018; Zheng, Li, and Duan 2019), and precast construction optimisation problems (Leu and Hwang 2001, 2002; Chan and Hu 2002; Benjaoran and Dawood 2003; Benjaoran, Dawood, and Hobbs 2005; Ko and Wang 2011; Chen, Yang, and Tai 2016; Yang, Ma, and Wu 2016). Compared to in-site concrete structures, precast concrete structures exhibit higher levels of production efficiency and showed more and are playing a more important role by employing highly effective manufacturing processes (Yang, Ma, and Wu 2016). Kong et al. (2017) considered the manufacture, transportation and on-site assembly sectors of precast construction projects, and proposed a dynamic programming algorithm and significantly reduced the construction waste. Chen et al. (2018) studied an automated guided vehicle (AGV)-based flow production system for the modular prefabricated horizon, and proposed a simulation based non-dominated sorting genetic algorithm. In precast production, a prefabricated building is manufactured and constructed via prefabrication from factory-made components transported and assembled on-site to form a complete building. Therefore, precast construction projects can generally be divided into three sectors, i.e. manufacture processing, transportation and on-site assembly (Yang, Ma, and Wu 2016).

Based on the above three construction processes, researchers have modelled this form of production based on flowshop scheduling problems (FSSPs) (Leu and Hwang 2001, 2002; Chan and Hu 2002; Benjaoran and Dawood 2003; Benjaoran, Dawood, and Hobbs 2005; Ko and Wang 2011; Chen, Yang, and Tai 2016; Yang, Ma, and Wu 2016; Li, Pan, and Duan 2016; Li, Duan et al. 2018) job shop scheduling problems (Arashpour et al. 2016), and flexible job shop scheduling problems (Anvari, Angeloudis, and Ochieng 2016). In addressing flowshop scheduling in a precast production system, Chan and Hu (2002) presented a genetic algorithm (GA) with a combination of two objective functions for minimising makespan and total tardiness penalties incurred from late product deliveries. Leu and Hwang (2001) modelled one of three precast working zones with the flowshop, and considered constrained resources including manpower, cranes, steam curing machines,

*Corresponding author. Email: lijunqing@lcu-cs.com; duanpeiyong@sdnu.edu.cn

reinforcement cage storage space, and steel forms. Further, Leu and Hwang (2002) investigated impacts of different product due dates on the makespan of the precast production process and addressed this using the GA algorithm. Benjaoran, Dawood, and Hobbs (2005) proposed a Bespoke Precast Flow Shop Scheduling Model (BP-FSSM) and optimised it by considering multi-objective GAs. Benjaoran and Dawood (2006) formulated six-step precast component production as a flowshop scheduling problem and utilised a GA-based optimisation algorithm to minimise the total flowtime. Ko and Wang (2011) considered precast production in considering resources and buffer sizes between stations and addressed this by using a multi-objective genetic algorithm to optimise minimum makespan and tardiness penalties. Yang, Ma, and Wu (2016) proposed a flowshop scheduling model based on multiple production lines for a precast production system and applied GA optimisation to minimise changes in types of precast components involved during production. Chen, Yang, and Tai (2016) performed process reengineering to create efficient precast processes. Besides the flowshop scheduling problem, Arashpour et al. (2016) considered an off-site precast production system by modelling it as a job shop scheduling problem (JSSP) to shorten changeover periods and thus to minimise wasted transitioning from one product class to another. Anvari, Angeloudis, and Ochieng (2016) investigated a multi-objective GA searching technique for minimising time and costs required while maximising safety levels and modelled a precast production system with a flexible job shop scheduling problem (FJSSP).

From the above literature we can see that many studies have modelled precast production as a scheduling problem while considering one workstation. However, distributed features of a prefabricated production system have not been widely examined. Currently, more and more precast components must be processed by more than one distributed workstation to increase production efficiency levels while decreasing waiting times and production costs (Naderi and Ruiz 2010, 2014; Lin, Ying, and Huang 2013; Wang, Wang et al. 2013; Xu et al. 2014; Fernandez-Viagas and Framinan 2015; Lin and Ying 2016; Rifai, Nguyen, and Dawal 2016; Wang, Huang, and Qin 2016; Bargaoui, Driss, and Ghédira 2017; Deng and Wang 2017; Komaki and Malakooti 2017; Lin, Wang, and Li 2017; Ying et al. 2017). Naderi and Ruiz (2010) characterised the distributed permutation flow-shop scheduling problem (DPFSP) and proposed six Mixed Integer Linear Programming (MILP) models and solved these models by employing simple factory assignment rules together with 14 heuristics. Wang, Wang et al. (2013) proposed an effective estimation of distribution algorithm (EDA) to solve the DPFSP. Lin, Ying, and Huang (2013) presented a modified iterated greedy (MIG) algorithm for this problem to minimise the maximum completion time for all factories involved. Naderi and Ruiz (2014) investigated a scatter search (SS) method along with restarts and local search methods for this problem to optimise makespan. Xu et al. (2014) developed an effective hybrid immune algorithm (HIA), wherein four search operators and crossover operators are embedded. Fernandez-Viagas and Framinan (2015) proposed a heuristic exploitation of the specific structure of the DPFSP problem. Wang, Huang, and Qin (2016) presented a fuzzy logic-based hybrid estimation of distribution algorithm (FL-HEDA) to address DPFSPs under conditions of machine breakdown with the makespan criterion. Rifai, Nguyen, and Dawal (2016) developed a novel multi-objective adaptive large neighbourhood search (MOALNS) algorithm to simultaneously satisfy three objectives of minimising makespan, total cost and average tardiness values in consideration of the reentrant characteristic of DPFSP. Lin and Ying (2016) solved the no-wait flowshop scheduling problem (DNFSP) by developing a mixed integer programming (MIP) mathematical model and an iterated cocktail greedy (ICG) algorithm. Bargaoui, Driss, and Ghédira (2017) investigated an artificial chemical reaction meta-heuristic to minimise the maximum completion time. Ying et al. (2017) presented an Iterated Reference Greedy (IRG) algorithm for the distributed no-idle permutation flowshop scheduling problem (DNIPFSP) with the objective of minimising the makespan. Komaki and Malakooti (2017) minimised the makespan of the distributed no-wait flow shop scheduling problem utilising a general variable neighbourhood search (GVNS) algorithm. Deng and Wang (2017) developed a competitive memetic algorithm (CMA) to solve the multi-objective DPFSP using the makespan and total tardiness criteria. Lin, Wang, and Li (2017) addressed the distributed assembly permutation flow-shop scheduling problem (DAPFSP) using a backtracking search hyper-heuristic (BS-HH) algorithm. Zhang, Xing, and Cao (2017) solved the distributed flowshop scheduling problem with flexible assembly and setup time using a constructive heuristic (TPHS) and two hybrid metaheuristics.

The artificial bee colony (ABC) algorithm was proposed by Karaboga (2005) and has been applied to solve many different types of problems such as those related to numerical function optimisation (Karaboga and Basturk 2007), lot-streaming flow shop scheduling (Pan et al. 2011), flexible job shop scheduling (Wang, Zhou, Xu, and Liu 2012, 2013; Wang, Zhou, Xu, Wang et al. 2012; Li, Pan, and Tasgetiren 2014), no-idle permutation flowshop scheduling (Tasgetiren et al. 2013), hybrid flow shop scheduling (Li and Pan 2015; Li, Sang et al. 2018), no-wait job shop scheduling (Sundar et al. 2017), parallel batch-processing machine scheduling (Zhang, Chang et al. 2017), steelmaking scheduling problems (Li et al. 2018), image steganalysis (Mohammadi and Abadeh 2014), transportation energy demand (Sonmez, Akgüngör, and Bektaş 2017), crowd evacuation in buildings (Liu et al. 2018), cloud manufacturing (Wang et al. 2015; Zhou and Yao 2017a, 2017b), pattern recognition (Joardar et al. 2017), and reverse logistics network systems (Li et al. 2017). In addition, many researchers have also developed different approaches for improving the performance of the classical ABC algorithm, such as multi-species co-evolution (Zhang, Liu, and Ding 2014), cooperative co-evolutionary ABC based on a hierarchical

communication model (Hu, Liu, and Zhang 2016), multi-objective ABC based on decomposition by PBI method (Bai and Liu 2016).

From the recent researches on the precast production system and the distributed flowshop scheduling problem, we find that: (1) there are several literatures considered solving the distributed flowshop scheduling problems, however, fewer of them have included realistic constraints such as distance index; (2) very few studies have addressed precast production as a DPFSP problem; and (3) the distributed production process can reduce production waste levels while considerably improving production efficiency levels, however, efficient optimisation algorithms are needed to consider both problem structures and objective features in this type of realistic DPFSP problem. Therefore, in this paper, we propose an artificial bee colony (ABC) algorithm as a means of addressing the distributed precast construction problem. The remainder of this paper is organised as follows. Section 2 describes the problem. Then, the proposed algorithm is shown in Section 3. Section 4 reports the experimental results and compares them with those of other algorithms presented in the literature to demonstrate the performance of the proposed algorithm. Finally, the last section presents conclusions drawn from our work.

2. Problem description

2.1. Description of the problem

In a typical precast concrete factory, the production routine involves several operations or processes such as mould prefabrication, reinforcement prefabrication, mould assembly, the placement of reinforcements and embedded parts, concrete mixing, casting, curing, mould stripping, product finishing and product stocking (Benjaoran, Dawood, and Hobbs 2005). All precast components or jobs must follow the same routine across distributed workstations over different or similar processing periods. Therefore, the precast production process can be modelled as a distributed flow shop scheduling problem (DFSSP) where there are n jobs and s parallel factories and where each factory employs m devices to complete the above production processes independently. First, each component should select a factory to complete its m operations. After factory selection, we must determine the processing sequence for each factory, and all precast components must follow the same processing sequence for the assigned factory. All precast components and devices are available at time zero. Preemption is not allowed in that any job cannot be interrupted before the completion of its current operations. In the prefabricated system, due to the large size of the component, it is not easy to transport such components from a factory located far away. Therefore, in this study, rather than exploring the canonical distributed flowshop scheduling problem, we consider the transport costs of the prefabricated system.

The aim of the production system is to schedule each job on each device to minimise the makespan value. The following assumptions are applied in this study:

- There are infinite buffers or storage between any two consecutive operations.
- Each device can process at most one operation at a time, and each operation should be processed on at most one machine at a time.
- The processing time of each operation should be a positive integer.
- Any two operations should not overlap on the same machine, i.e. the start time of the successor job must be greater than the completion time of its immediately predecessor job plus the setup time between them.
- The starting time of any operation should be greater or equal to its release time from the previous stage.
- All operations are strictly assigned to one machine at each stage.
- Any operation has exactly one immediate processor or successor operation for the same machine.
- There are enough resources for processing precast components, including manpower or workers, steaming curing capacities, cranes, reinforcement cage storage spaces, and steel forms.
- Different factories employ different distance indexes for each job and the processing time includes this type of factor. In other words, we include the distance cost in the processing time of each job for each factory.

The notations used in this paper are summarised as follows:

Indices

i	index of precast components or jobs, $i = 1,2, \ldots ,n$;
j	index of devices or machines, $j = 1,2, \ldots ,m$;
f	index of factories, $f = 1,2, \ldots ,s$.

Parameters

n	total number of precast components or jobs
m	total number of machines
s	total number of factories

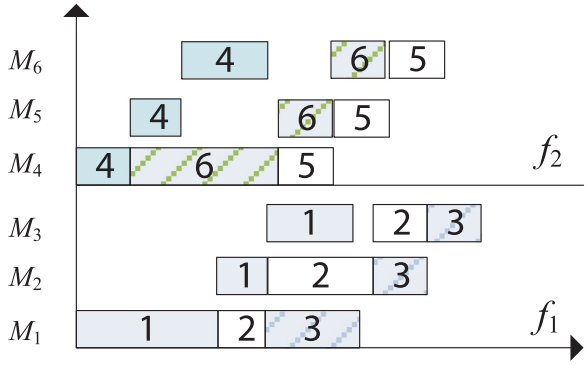


Figure 1. An example of a Gantt chart.

$O_{i,j}$	the jth operation of job i
$DI_{i,f}$	distance index of job i in factory f
$p_{i,j,f}$	The processing time requirement of job i carried out on machine j in factory f
$E_{f,j,i}$	The earliest completion time of $O_{i,j}$ in factory f
$c^f_{\max}$	The maximum completion time of all operations in factory f
$c_{\max}$	The maximum completion time of all operations in all factories
$T_{f,j,i}$	The duration between the starting time of $O_{i,j}$ and the end of the operations in factory f
J	set of n jobs, $J = \{J_1, J_2, \ldots, J_n\}$
M	set of m machines, $M = \{M_1, M_2, \ldots, M_m\}$
Decision variables	
$s_{i,j}$	starting time of job i at stage j;
$c_{i,j}$	completion time of job i at stage j;
$x_{i,f}$	a 0/1 variable that is equal to one if and only if job J_i has been assigned to be processed at factory f.

The aim of this study is to minimise the makespan:

$$c_{\max} = \max\,(c^f_{\max}), \quad f = 1, 2, \ldots, s \tag{1}$$

2.2. Example

Figure 1 presents a distributed precast production example where there are six jobs or precast components to be processed, two factories, and three machines in each factory. Each precast component must be processed on three production lines according to the determined sequence (from the first to the last machine in the selected factory). Figure 1 shows that three jobs are to be processed at the first factory, i.e. J_1, J_2, and J_3, whereas the other three jobs (J_4, J_5, and J_6) are to be processed at the second factory. The figure shows that the two groups of precast components can be processed in parallel, and thus the production time can be decreased dramatically.

3. The proposed algorithm

3.1. Solution representation

The classical solution representation of the flow shop scheduling problem is generally executed using the permutation based representation, where each solution is represented by a string of integers and each integer represents a job number. However, to solve the distributed precast construction problem, the permutation representation must not have enough information to describe the assigned factory.

In the canonical PFSP, general representation involves employing the permutation-based coding method, where each solution is represented a string of integers and each integer is the job number. Jobs are individually scheduled according to their ordering in the solution representation. A common way to assign a factory a scheduling job involves assigning a random factory to it. Hereafter we denote this form of solution representation as SR-I. Another form of solution representation involves two-vector-based representation as used in FJSPs (flexible job shop scheduling problems) or HFSPs (hybrid flowshop scheduling problems), and this involves two vectors, i.e. the factory vector and the scheduling vector. The factory vector records the factory number assigned to each job, and the scheduling vector records the processing sequence for each

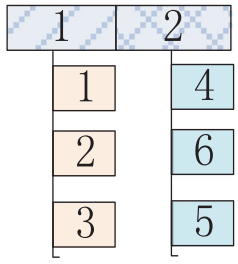


Figure 2. Example of coding representation via SR-III.

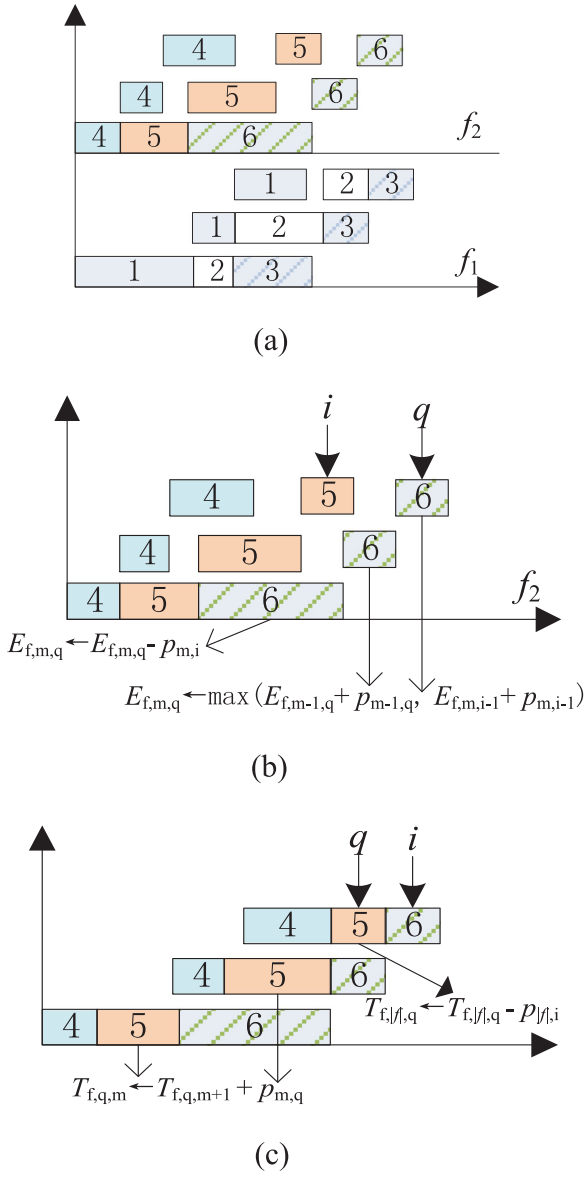


Figure 3. Gantt chart for the destruction method. (a) the previous Gantt char, (b) *i* is not the last one in *f*, (c) *i* is the last one in *f*.

job. The lengths of the two vectors are set to the number of jobs. Hereafter we denote this form of solution representation as SR-II. However, we found that in using SR-II, it is difficult to design the mutation operator. The third type takes a two-dimensional vector as the representation (Naderi and Ruiz 2010) denoted as SR-III. In the two-dimensional vector, it is generally to set a vector for each factory to store the job processing order of each factory. For the given example illustrated in Figure 1, the code representation is described in Figure 2 where each factory involves a job processing vector. For example,

Algorithm 1: Destruction method

input: factory f and position i

output: the makespan after removing the job in i: c_{max}

if i is not the last position in f **then** //(as illustrated in Fig. 3(b))
  $q \leftarrow i+1$ //q is the position used to compute the makespan
  For each machine m from $0 \rightarrow |f|$ in f **do** //$|f|$ is the number of machine in f
    if m=0 **then**
      $E_{f,m,q} \leftarrow E_{f,m,q} - p_{m,I}$ //(as illustrated in Fig. 3(b))
    otherwise
      if $i \neq 0$ **then** //(as illustrated in Fig. 3(b))
        $E_{f,m,q} \leftarrow \max(E_{f,m-1,q} + p_{m-1,q}, E_{f,m,i-1} + p_{m,i-1})$
      otherwise
        $E_{f,m,q} \leftarrow E_{f,m-1,q} + p_{m-1,q}$
      end
    end
  end
otherwise// if i is the last one in f
  $q \leftarrow i-1$
  $T_{f,|f|,q} \leftarrow T_{f,|f|,q} - p_{|f|,i}$ //(as illustrated in Fig. 3(c))
  For each machine m from $|f|-1 \rightarrow 0$ in f **do**
    $T_{f,q,m} \leftarrow T_{f,q,m+1} + p_{m,q}$ //(as illustrated in Fig. 3(c))
  end
For each machine m from $0 \rightarrow |f|$ in f **do**
  $c_{max} = \max(c_{max}, E_{f,m,q} + T_{f,m,q})$
end

for the first factory, the job processing sequence is J_1, J_2, and J_3. The comparative results and analysis for the solution representation verify that the proposed SR-III performs better than the other two coding methods.

3.2. Decoding method

For the given solution representation applying SR-III, the decoding process involved is very simple. To decode for each factory, we simply process each job according to its sequence in the coding representation. When scheduling each job to each machine for each factory, operations should be initiated after the completion of previous operations and the processing machine should also be idle during this period. The entire process is generally similar to the process involved in scheduling the canonical permutation flow shop scheduling problem. The processes differ in that the problem considered in this study should consider the factory assigned to each job. For the example shown in Figure 2, its decoding Gantt chart is given in Figure 1.

3.3. Distributed iterated greedy method

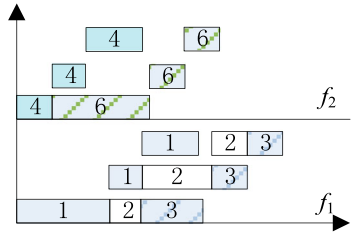
In this section, we propose a distributed iterated greedy (DIG) approach for solving flow shop scheduling problems under a distributed environment.

3.3.1. Destruction method

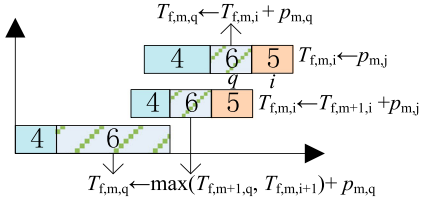
For distributed flowshop scheduling problems, several factories are set in the parallel model. Therefore, removing several jobs from one factory may not optimise the makespan of the whole system, as the other factories may determine the makespan of the system. Therefore, after removing one or several jobs from one factory, we must determine whether the makespan objective has been affected by the above process. When the makespan objective has improved by removing the selected job, we can process the construction method for further processing; otherwise, it will be valueless to perform the following steps. The time complexity of the destruction process is $O(snm)$. The destruction method is given as follows.

3.3.2. Construction method

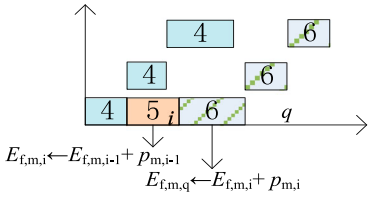
After the destruction process, DIG performs the construction method to find the optimal position for each deleted operation. The detailed steps of the construction method are given in Algorithm 2. It is obvious that the time complexity of the construction process is $O(sn^2m)$.



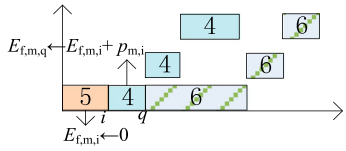
(a)



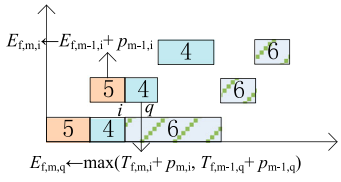
(b)



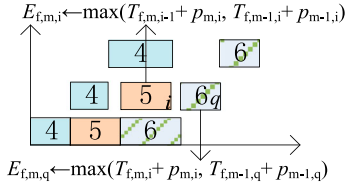
(c)



(d)



(e)



(f)

Figure 4. Gantt chart for the construction method. (a) the previous Gantt char, (b) i is positioned behind the current f, (c) $i \neq 0$ and $m = 0$, (d) $i = 0$ and $m = 0$, (e) $i = 0$ and $m \neq 0$, (f) $i \neq 0$ and $m \neq 0$

Algorithm 2: Construction method

input: factory f, position i, and job j
output: the makespan after inserting the job in position i: c_{max}

if i is positioned behind the current f **then** //(as illustrated in Fig. 4(b))
  $q \leftarrow i$-1
  For each machine m from $|f| \rightarrow 0$ in f **do** //$|f|$ is the number of machine in f
    if m=$|f|$ **then**
      $T_{f,m,i} \leftarrow p_{m,j}$
      $T_{f,m,q} \leftarrow T_{f,m,i} + p_{m,q}$
    otherwise
      $T_{f,m,i} \leftarrow T_{f,m+1,i} + p_{m,j}$
      $T_{f,m,q} \leftarrow \max(T_{f,m+1,q}, T_{f,m,i+1}) + p_{m,q}$
    end
otherwise// if i is from 0 to $|f|$ in f
  $q \leftarrow i$+1
  For each machine m from $0 \rightarrow |f|$ in f **do**
    if m=0 **then**
      if i=0 **then**
        $E_{f,m,i} \leftarrow 0$ //(as illustrated in Fig. 4(d))
      otherwise
        $E_{f,m,i} \leftarrow E_{f,m,i-1} + p_{m,i-1}$ //(as illustrated in Fig. 4(c))
      end
      $E_{f,m,q} \leftarrow E_{f,m,i} + p_{m,i}$ //(as illustrated in Fig. 4(c))
    otherwise
      if i=0 **then**
        $E_{f,m,i} \leftarrow E_{f,m-1,i} + p_{m-1,i}$ //(as illustrated in Fig. 4(e))
      otherwise
        $E_{f,m,i} \leftarrow \max(T_{f,m,i-1} + p_{m,i}, T_{f,m-1,i} + p_{m-1,i})$ //(as illustrated in Fig. 4(f))
      end
      $E_{f,m,q} \leftarrow \max(T_{f,m,i} + p_{m,i}, T_{f,m-1,q} + p_{m-1,q})$
    end
  end
  For each machine m from $0 \rightarrow |f|$ in f **do**
    $c_{max,f} = \max(c_{max,f}, E_{f,m,q} + T_{f,m,q})$
  end
  $c_{max} = \max_{\exists f} c_{max,f}$

3.4. Initialisation of the population

The main task of the DFSP is to assign a factory for each job. In this section we present two factory assignment methods, which are given as follows. (1) A random factory assignment rule denoted as RFA that assigns a random selected factory to each job and (2) an average factory workload assignment rule denoted as AFWA that first computes the workload for each factory and which then assigns the factory with the current minimum workload. Specifications of the AFWA rule are given in Algorithm 3.

Algorithm 3: AFWA rule

input: factory f and the processing time for each job.
output: the assigned factory for each job

for each job i **do**
  record the total processing time of job i in the vector *job_totalTime*
end
randomly sort each job and store them in the sorted vector.
for each job i in the sorted vector **do**
  count the current workload for each factory.
  select the factory with the lowest workload f for job i.
  append job i into the solution representation vector corresponding to factory f.
  add the total processing time of job i to the workload of factory f.
end

It is evident that by using the proposed AFWA rule, a factory can be assigned to the jobs with similar workloads, and therefore, the maximum completion time of each factory may be similar. As the makespan of the considered problem is the maximum completion level for each factory, when using the average workload method the whole makespan criterion tends to be superior to the random rule. Experiment comparisons also verify this conclusion.

In addition, the initial population quality level is crucial to the proposed algorithm. An initial population with a high level of quality and diversity may result in a faster coverage of good solutions. In this study we employ a simple initialisation approach as follows. (1) P_{size}-1 solutions are generated in a random manner using the AFWA rule where P_{size} is the population size of the system, and (2) one solution is generated by using the proposed DIG approach after applying the AFWA rule to start scheduling each job in each factory once again.

3.5. Local search operators

Local search operators are designed differently according to different coding representations. For SR-I, it is common to use swap or insert operators to generate a different job sequence. SR-II unlike SR-I should consider a difference in factory vectors by applying the following rules: (1) randomly change a factory for a randomly selected operation and (2) randomly swap two factories for two randomly selected jobs. These two factory mutation rules can be used in a random manner.

It should be noted that in SR-II, permutations in the factory and scheduling vectors are independent. Therefore, when we switch a factory for a job, it is difficult to determine the right position in the new factory to execute it. In this section, we design a fast job insertion method referred to as *fast_job_insertion* described in detail in Algorithm 4, which has following advantages: (1) the method considers assigning a factory and scheduling job sequence simultaneously, and therefore can improve the solution performance; (2) it has a fast speed to find the optimal position for each swapped job, and the time complexity of the job insertion process is $O(snm)$; and (3) with the fast insertion method, many types of local search operators can be designed considering the problem features.

Using the fast job insertion method we develop three types of local search operators as follows: (1) the LSO-I method, which randomly changes a factory for a randomly selected operation and which finds an optimal position for the inserting operation via the *fast_job_insertion* method and (2) the LSO-II method, which randomly swaps two factories for two randomly selected jobs and which finds an optimal position for the two inserting operations using the *fast_job_insertion* method. It should be noted that when we test the above two local search operators we find that the LSO-II method performs better than the former method.

3.6. Improved artificial bee colony algorithm

In this section, we develop an improved artificial bee colony algorithm referred to as IABC that involves an employee bee, onlooker bee, scout bee and the exploration approach.

3.6.1. Employee bee operator

The employee bee is to perform the exploitation process based on the given solution. As in (Li, Pan, and Gao 2011; Li and Pan 2015; Li et al. 2017), the employee bee of the proposed IABC algorithm also performs the local search operator discussed in sub-section 3.4. Steps of the proposed employee bee operator are given as follows.

Step 1. For each solution s_i in the current population, perform the following steps.
Step 2. For given solution s_i, perform the local search operator *fast_job_insertion* described in Algorithm 4 to generate a neighbouring solution s_x.
Step 3. Evaluate the neighbouring solution s_x. Perform the following steps: (1) if the neighbouring solution is better than the current solution, replace the latter; (2) if the neighbouring solution s_x is worse than the current solution, ignore it; and (3) if s_x is better than the global best solution found thus far, replace the latter to complete the elicit process.

3.6.2. Onlooker bee operator

The onlooker bee is to perform a further exploitation process around the selected better solution. In this section, the onlooker bee is performed as follows.

Step 1. Perform the following steps P_{size} times.
Step 2. Randomly select two solutions from the current population and select the better solution as the current onlooker bee.

Algorithm 4: *fast_job_insertion*

input: factory f and the inserted job i.
output: solution representation after inserting job i into factory f

compute the earliest start and tail times for each job processed in factory f not including job i
for each processing position p in factory f **do**
  if p is the last processing position in factory f **then**
    for m from $|f| \rightarrow 0$ **do** //$|f|$ is the number of machines in f
      if m is the last machine in factory f **then**
        let $pt_{i,m}$ be the processing time of job i
        let $pt_{p-1,m}$ be the processing time of job in position p -1
        $T_{m,p} \leftarrow pt_{i,m}$ // $T_{m,p}$ is the tail time of job in position p executed onmachine m
        $T_{m,p-1} \leftarrow T_{m,p} + pt_{p-1,m}$
      otherwise
        $T_{m,p} \leftarrow T_{m+1,p} + pt_{i,m}$
        $T_{m,p-1} \leftarrow \max\{ T_{m+1,p-1}, T_{m,p}\} + pt_{p-1,m}$
      end
    end
    $cp \leftarrow p$-1 //cp is the computing position for the makespan
  otherwise
    $cp \leftarrow p$+1
    for m from $0 \rightarrow |f|$ **do**
      if m equals 0 **then**
        if p equals 0 **then** $E_{m,p} \leftarrow 0$
        otherwise $E_{m,p} \leftarrow E_{m,p-1} + pt_{p-1,m}$
        end
        $E_{m,p+1} \leftarrow E_{m,p} + pt_{i,m}$
      otherwise
        if p equals 0 **then** $E_{m,p} \leftarrow E_{m-1,p} + pt_{i,m-1}$
        otherwise $E_{m,p} \leftarrow \max\{ E_{m-1,p} + pt_{i,m-1}, E_{m,p-1} + pt_{p-1,m} \}$
        end
        $E_{m,p+1} \leftarrow \max\{ E_{m,p} + pt_{i,m}, E_{m-1,p+1} + pt_{p+1,m-1} \}$
      end
    end
  end
end

Step 3. Generate a neighbouring solution around the current onlooker bee by performing the local search operator *fast_job_insertion* described in Algorithm 4.
Step 4. Evaluate the newly-generated neighbouring solution and use it to try to replace the current onlooker bee and the global best solution; the replace process is similar to the process described in sub-section 3.5.1.

3.6.3. Scout bee operator

If a given solution has not been improved after a given number of iterations, it should be replaced with a more promising solution. Common methods used may be classified into two categories. The first approach to improvement involves replacing the unimproved solution with a randomly generated solution, and this method is commonly applied when using the canonical ABC algorithm. The second approach to improvement involves replacing the unimproved solution with a permutation of the global best solution, as the global best solution always includes promising information (Pan, Li). In this study we developed an improved scout bee operator as follows.

Step 1. Record the solutions that have not been improved after L_t iteration times into a vector referred to as *LS*.
Step 2. Randomly select a solution from vector *LS* as the current scout bee.

Step 3. Perform the local search operator *fast_job_insertion* described in Algorithm 4 *SN* times on the global best solution found thus far to generate a neighbouring solution S_x.
Step 4. Evaluate the newly generated neighbouring solution S_x and use it to directly replace the current scout bee. Try to use S_x to replace the global best solution if the former is better than the latter.

3.7. The framework of the proposed algorithm

The detailed steps of the proposed IABC algorithm are as follows:

Step 1: Initialisation phase.

Step 1.1 Set the system parameters.
Step 1.2 Initialise the population discussed in sub-section 3.4.

Step 2: If the stopping criterion is satisfied, output the best solution; otherwise, perform steps 3–5.
Step 3: Exploitation search phase.

Step 3.1 Apply the employee bee operator discussed in sub-section 3.5.1.
Step 3.2 Apply the onlooker bee operator discussed in sub-section 3.5.2.
Step 3.3 Record the best solution found thus far.

Step 4: Exploration search phase.

Step 4.1 Apply the scout bee operator discussed in sub-section 3.5.3.
Step 4.2 Record the best solution found thus far.

Step 5: Go back to step 3.

4. Experimental results

This section discusses computational experiments conducted to evaluate the performance of the proposed algorithm. Our algorithm was applied in C++ on an Intel Core i7 3.4 GHz PC with 16 GB memory. To test the performance of the proposed algorithm, we select 720 large-scale instances from a website (http://soa.iti.es/rruiz) where the number of jobs is $n = \{20, 50, 100, 200, 500\}$, the number of machines is $m = \{5, 10, 20\}$, and the number of factories is $f = \{2, 3, 4, 5, 6, 7\}$. To render the instances suited to the prefabricated system, we extend these 720 instances by considering the distance index, which is a real number generated from the uniform distribution [0.5, 1.5]. After considering the distance index, the final processing time of each job is computed as follows:

$$p'_{i,j,f} = \lfloor p_{i,j,f} \times DI_{i,f} \rfloor \tag{2}$$

where $p'_{i,j,f}$ is the final processing time required to consider the distance cost index ($DI_{i,f}$) and where $p_{i,j,f}$ is the processing time provided on the website (http://soa.iti.es/rruiz). The notation $\lfloor \cdot \rfloor$ is used to transform a real number into an integer by maintaining the integral part. For example, we use $\lfloor 4.5 \rfloor$ to achieve a processing time of 4.

The relative percentage increase (RPI) is used as a performance measure and is computed as follows:

$$\mathrm{RPI(C)} = \frac{C_c - C_b}{C_b} \times 100 \tag{3}$$

where C_b is the best makespan found from the compared algorithms while C_c is the best solution obtained from the given algorithm.

4.1. Setting parameters

Each instance can be characterised by the following parameters: the number of precast components or jobs (n), the number of devices or machines (m), and the number of factories (s).

System parameters include the number of iterations during which the solution does not improve (L_t), the population size (P_{size}) and the size of neighbours during the scout bee search process (SN). The levels of the three parameters are given in Table 1. The DOE Taguchi method (Montgomery 2005) is used to test the influence of these three parameters on the performance of the proposed algorithm. As the three parameters are set with four factor levels, an orthogonal array

Table 1. Combinations of parameter values.

	Level			
Parameter	1	2	3	4
PS	10	30	50	100
SN	1	3	5	10
L_t	5	10	15	20

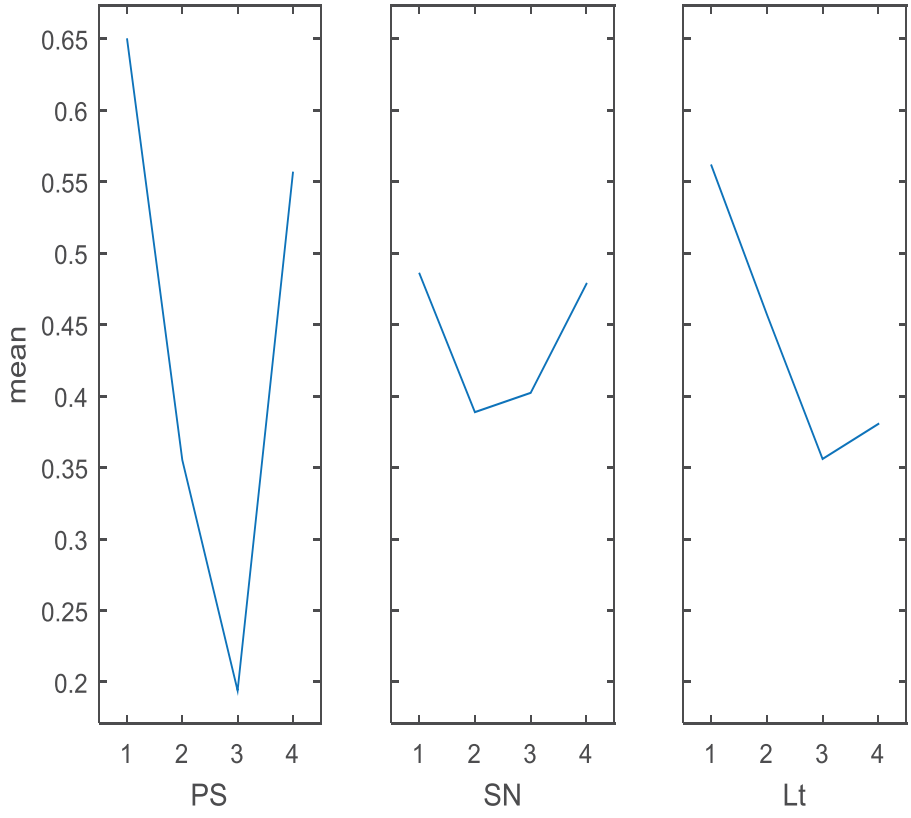


Figure 5. Factor level trends of the three key parameters.

$L_{16}(4^3)$ is used. For each parameter combination, the proposed algorithm is independently run 30 times, and then the average makespan value obtained from the proposed algorithm is collected as the response variable (RV). Table 2 gives the response values of different combinations of these three parameters. Figure 5 reports the factor level trends of the two parameters while Table 4 lists the average response value for each level of the considered parameters.

Figure 5 shows that the proposed algorithm performs better under three parameters of the following levels: *PS* of level 3, *SN* of level 2, and L_t of level 3. We can also see from Figue 5 that the parameter *PS* is more critical than the other two parameters of the proposed algorithm. A large *PS* value denotes that more computational resources are consumed through the exploration procedure and that the algorithm will lose exploitation capacity. A *PS* value that is too small denotes a loss of algorithm exploration ability. Therefore, to balance exploration and exploitation capacity, the suitable value of the key parameter *PS* is set as 50 in the proposed algorithm. According to the above analysis, suitable values for the three considered parameters are 50, 3 and 15 for *PS*, *SN* and L_t, respectively.

4.2. Efficiency of the proposed components

This section gives the experiment results of the proposed components, namely the coding method, DIG method, local search approach, and scout bee heuristic.

Table 2. Response values of different combinations of parameters.

Experiment number	Factor			RV
	PS	*SN*	L_t	
1	1	1	1	0.98
2	1	2	2	0.55
3	1	3	3	0.435
4	1	4	4	0.636
5	2	1	2	0.41
6	2	2	1	0.348
7	2	3	4	0.302
8	2	4	3	0.361
9	3	1	3	0.083
10	3	2	4	0.112
11	3	3	1	0.292
12	3	4	2	0.289
13	4	1	4	0.472
14	4	2	3	0.545
15	4	3	2	0.58
16	4	4	1	0.628

Table 3. Comparisons of the proposed local search operator.

	RPI			time (*s*)		
	SR-I	SR-II	IABC	SR-I	SR-II	IABC
20–5	0.11	0.57	**0.00**	1.55	0.53	1.12
20–10	**0.00**	0.73	0.12	2.97	0.97	2.48
20–20	**0.00**	2.33	1.40	5.28	2.15	5.20
50–5	1.34	0.39	**0.00**	5.67	4.55	4.90
50–10	1.22	0.42	**0.00**	11.80	9.33	11.30
50–20	1.26	0.66	**0.00**	25.13	15.60	22.58
100–5	2.41	0.70	**0.00**	10.97	10.93	12.10
100–10	2.20	0.79	**0.00**	23.17	22.22	23.95
100–20	2.05	0.76	**0.00**	47.90	36.35	49.55
200–10	2.66	0.92	**0.00**	40.55	49.80	51.87
200–20	2.55	1.04	**0.00**	92.62	91.27	105.33
500–20	1.94	0.76	**0.00**	196.50	240.70	292.03
mean	1.48	0.84	**0.13**	38.68	40.37	48.53

Note: Best values are in bold.

4.2.1. Efficiency of the proposed solution representation method

To investigate the effectiveness of the proposed local search discussed in sub-section 3.1, we tested the three compared algorithms using three different solution representation methods and namely, SR-I, SR-II and IABC where IABC embeds the SR-III operator. The other components of the three compared algorithms are set at the same values. The three algorithms compared are tested on the same PC and with the same test instances. After 30 independent runs, the results for each instance are collected to draw comparisons, which are given in Table 3.

In Table 3, the first column provides the problem scale, which includes the number of jobs and machines involved in the system. The following three columns report RPI values obtained by SR-I, SR-II and IABC. The last three columns give the time consumed by the three algorithms. It can be observed from Table 3 that (1) IABC obtained 10 optimal values out of the 12 given types of instances, whereas SR-I obtained two optimal values and SR-II with no optimal values. (2) From the last row in the table, we can see that IABC clearly performs better than the other two algorithms compared.

To determine whether significant differences exist between the compared algorithms, we performed a multifactor analysis of variance (ANOVA) where the three compared algorithms are considered as factors. Figure 6(a) illustrates the means and 95% LSD (Fisher's Least Significant Difference) interval for the RPI values of the three compared algorithms. We can see that the *p*-value is close to zero, revealing significant differences between the three algorithms compared.

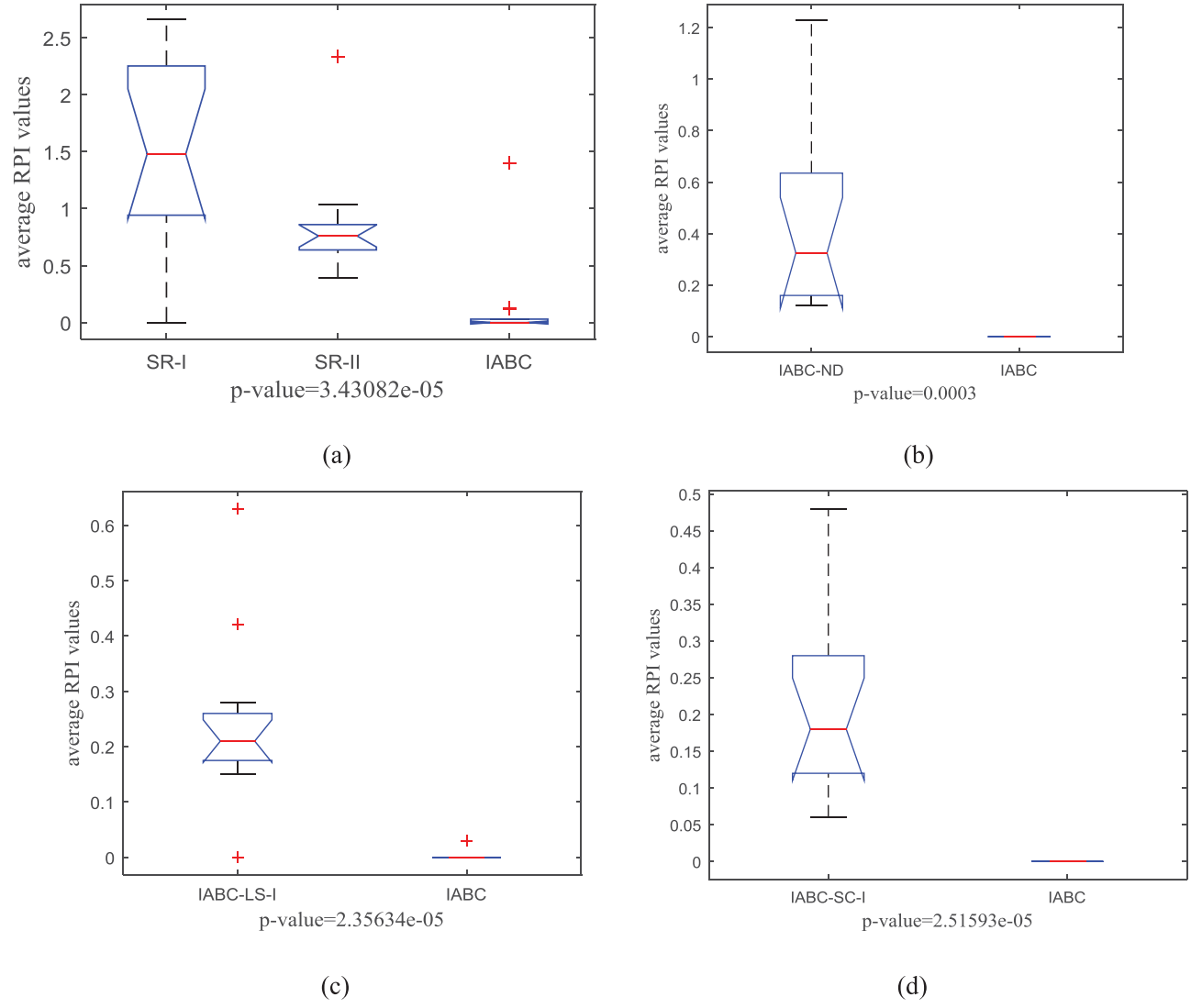


Figure 6. Means and 95% LSD interval for pairs of compared algorithms. (a) Comparisons of the solution representation methods, (b) Comparisons of the DIG methods, (c) Comparisons of the local search methods, (d) Comparisons of the scout bee methods.

4.2.2. Efficiency of the proposed DIG method

To investigate the effectiveness of the proposed DIG heuristic discussed in sub-section 3.3, we tested the compared algorithms and namely, IABC-ND and IABC. The other components of IABC-ND and IABC are set as the same except for the DIG heuristic when attempting to find the optimal position in the new factory for the re-assignment operation. The two algorithms compared are tested on the same PC and with the same test instances. After 30 independent runs, the results for each instance are collected to draw comparisons, which are given in Table 4.

Table 4 shows that (1) IABC obtained all optimal values of the given 12 types of instances and (2) from the last row in the table we can see that IABC clearly performs better than IABC-ND. Figure 6(c) illustrates the means and 95% LSD interval for RPI values for the two pairs of algorithms compared. We can see that the *p*-value is close to zero, revealing significant differences between the two pairs of algorithms compared. The main advantages of the proposed IABC method lie in the fact that when attempting to find the optimal position in the new factory for the re-assignment operation, the proposed algorithm uses the fast DIG method to compute the fitness of the obtained neighbouring solution (which applies time complexity $O(sn^2m)$) while the IABC-ND algorithm should consume $O(sn^3m)$ to compute the fitness of all possible insertion positions.

Table 4. Comparisons of the proposed DIG method.

	RPI		time (*s*)	
	IABC-ND	IABC	IABC-ND	IABC
20–5	0.33	**0.00**	0.58	1.12
20–10	0.18	**0.00**	1.40	2.48
20–20	0.14	**0.00**	2.73	5.20
50–5	0.20	**0.00**	4.95	4.90
50–10	0.12	**0.00**	9.07	11.30
50–20	0.12	**0.00**	19.28	22.58
100–5	0.51	**0.00**	13.37	12.10
100–10	0.43	**0.00**	26.88	23.95
100–20	0.32	**0.00**	52.20	49.55
200–10	0.90	**0.00**	58.63	51.87
200–20	0.76	**0.00**	116.70	105.33
500–20	1.23	**0.00**	301.15	292.03
mean	0.44	**0.00**	50.58	48.53

Note: Best values are in bold.

Table 5. Comparisons of the proposed local search operator.

	RPI		time (*s*)	
	IABC-LS-I	IABC	IABC-LS-I	IABC
20–5	0.63	**0.00**	1.43	1.12
20–10	0.42	**0.00**	3.28	2.48
20–20	**0.00**	0.03	6.50	5.20
50–5	0.28	**0.00**	5.43	4.90
50–10	0.21	**0.00**	11.18	11.30
50–20	0.24	**0.00**	24.33	22.58
100–5	0.21	**0.00**	12.63	12.10
100–10	0.20	**0.00**	26.25	23.95
100–20	0.21	**0.00**	51.37	49.55
200–10	0.17	**0.00**	55.75	51.87
200–20	0.18	**0.00**	108.48	105.33
500–20	0.15	**0.00**	292.68	292.03
mean	0.24	**0.00**	49.94	48.53

Note: Best values are in bold.

4.2.3. Efficiency of the proposed local search method

To investigate the effectiveness of the proposed local search discussed in sub-section 3.4, we tested the two local search operators (LS-I and LS-II), and the compared algorithms embedded in the two local search operators are denoted as IABC-LS-I and IABC, respectively. The other components of IABC-LS-I and IABC are set as identical. The two algorithms compared are tested on the same PC and with the same test instances. After 30 independent runs, the results for each instance are collected to draw comparisons, which are given in Table 5.

It can be observed from Table 5 that (1) IABC obtained 11 optimal values of the given 12 types of instances, whereas the IABC-LS-I obtains one optimal value. (2) From the last row in the table we can see that IABC clearly performs better than IABC-LS-I. Figure 6(c) illustrates the means and the 95% LSD interval of RPI values for the two pairs of algorithms compared. We can see that the *p*-value is close to zero, revealing significant differences between the two pairs of algorithms compared.

4.2.4. Efficiency of the proposed scout bee method

To investigate the effectiveness of the proposed scout bee method discussed in sub-section 3.5.3, we tested the two scout bee methods (IABC-SC-I and IABC). The two methods differ in that IABC-SC-I utilises the common scout bee method, i.e. to select a randomly generated solution to replace the current scout bee, whereas IABC embeds the scout bee operator

Table 6. Comparisons of the proposed scout bee heuristic.

	RPI		time (*s*)	
	IABC-SC-I	IABC	IABC-SC-I	IABC
20–5	0.46	**0.00**	0.35	1.12
20–10	0.48	**0.00**	0.77	2.48
20–20	0.46	**0.00**	1.27	5.20
50–5	0.19	**0.00**	2.63	4.90
50–10	0.13	**0.00**	4.47	11.30
50–20	0.18	**0.00**	8.78	22.58
100–5	0.18	**0.00**	7.78	12.10
100–10	0.14	**0.00**	15.72	23.95
100–20	0.09	**0.00**	30.28	49.55
200–10	0.14	**0.00**	41.15	51.87
200–20	0.08	**0.00**	88.35	105.33
500–20	0.06	**0.00**	260.45	292.03
mean	0.22	**0.00**	38.50	48.53

Note: Best values are in bold.

Table 7. Comparisons of the four algorithms (with CPU = 1).

	RPI				time (*s*)			
	VND	BSA	CMA	IABC	VND	BSA	CMA	IABC
20–5	0.36	0.30	0.31	**0.22**	0.00	0.00	0.00	0.00
20–10	0.16	0.12	0.12	**0.11**	0.11	0.14	0.13	0.14
20–20	0.08	**0.04**	**0.04**	0.06	0.29	0.34	0.36	0.38
50–5	0.20	0.16	0.15	**0.13**	0.19	0.20	0.20	0.20
50–10	0.18	**0.09**	0.10	**0.09**	0.56	0.60	0.59	0.64
50–20	0.16	**0.07**	**0.07**	0.08	1.27	1.37	1.40	1.41
100–5	0.20	0.12	0.12	**0.10**	0.63	0.61	0.61	0.64
100–10	0.20	0.12	0.12	**0.10**	1.36	1.38	1.40	1.44
100–20	0.19	0.11	0.11	**0.09**	2.84	2.77	2.86	2.91
200–10	0.21	0.13	0.12	**0.10**	2.92	2.84	2.82	2.90
200–20	0.20	0.12	0.12	**0.10**	6.00	5.64	5.76	5.89
500–20	0.17	0.11	0.10	**0.10**	22.11	14.51	14.42	14.97
mean	0.19	0.12	0.12	**0.11**	3.19	2.53	2.55	2.63

Note: Best values are in bold.

discussed in sub-section 3.5.3. The other components of the two compared algorithms are set as identical. The two algorithms compared are tested on the same PC and with the same test instances. After 30 independent runs, the results for each instance are collected to draw comparisons, which are given in Table 6.

It can be observed from Table 6 that (1) IABC obtained all optimal values of the given 12 types of instances. (2) From the last row in the table we can see that IABC clearly performs better than IABC-SC-I. Figure 6(d) illustrates the means and the 95% LSD interval for RPI values of the two pairs of algorithms compared. We can see that the *p*-value is close to zero, revealing significant differences between the two pairs of algorithms compared. The main advantage of the proposed scout bee method lies in the fact that it replaces the current solution with an individual with more promising information and therefore improves searching performance outcomes.

4.3. Comparisons with efficient algorithms

To draw a fair comparison between the proposed algorithm and the other three algorithms compared, i.e. VND (Variable Neighbourhood Descent, Naderi and Ruiz 2010), CMA (Competitive Memetic Algorithm, Deng and Wang 2017), and BSA (Backtracking Search Algorithm, Lin, Wang, and Li 2017), we code the above three compared algorithms and run them on the same PC under the same stop conditions. For each algorithm compared, solution representation, permutation operator and evolutionary approaches are utilised as shown in corresponding references. Each algorithm compared is independently run 30 times for each given instance. All algorithms adopt the same maximum elapsed CPU time limit of $t = n \times m \times$ CPU

Table 8. Comparisons of the four algorithms (with CPU = 10).

	RPI				time (*s*)			
	VND	BSA	CMA	IABC	VND	BSA	CMA	IABC
20–5	0.07	**0.01**	**0.01**	**0.01**	0.64	0.73	0.79	0.94
20–10	0.04	0.00	**0.00**	0.01	1.41	1.25	1.40	1.79
20–20	0.03	0.00	**0.00**	0.01	2.00	1.62	1.80	2.91
50–5	0.12	0.02	0.02	**0.00**	2.46	2.96	3.19	3.05
50–10	0.12	0.02	0.01	**0.00**	4.58	6.29	6.65	5.80
50–20	0.11	0.01	**0.00**	0.01	10.83	11.64	12.41	12.43
100–5	0.14	0.04	0.04	**0.00**	6.05	6.41	6.70	20.60
100–10	0.14	0.04	0.04	**0.00**	12.81	13.51	13.78	13.71
100–20	0.14	0.03	0.02	**0.00**	25.83	25.85	27.32	27.10
200–10	0.15	0.06	0.05	**0.00**	28.64	27.10	27.93	32.47
200–20	0.16	0.06	0.04	**0.00**	57.10	55.31	58.03	53.29
500–20	0.15	0.05	0.05	**0.00**	144.33	142.24	144.80	159.83
mean	0.11	0.03	0.02	**0.00**	24.72	24.57	25.40	27.83

Note: Best values are in bold.

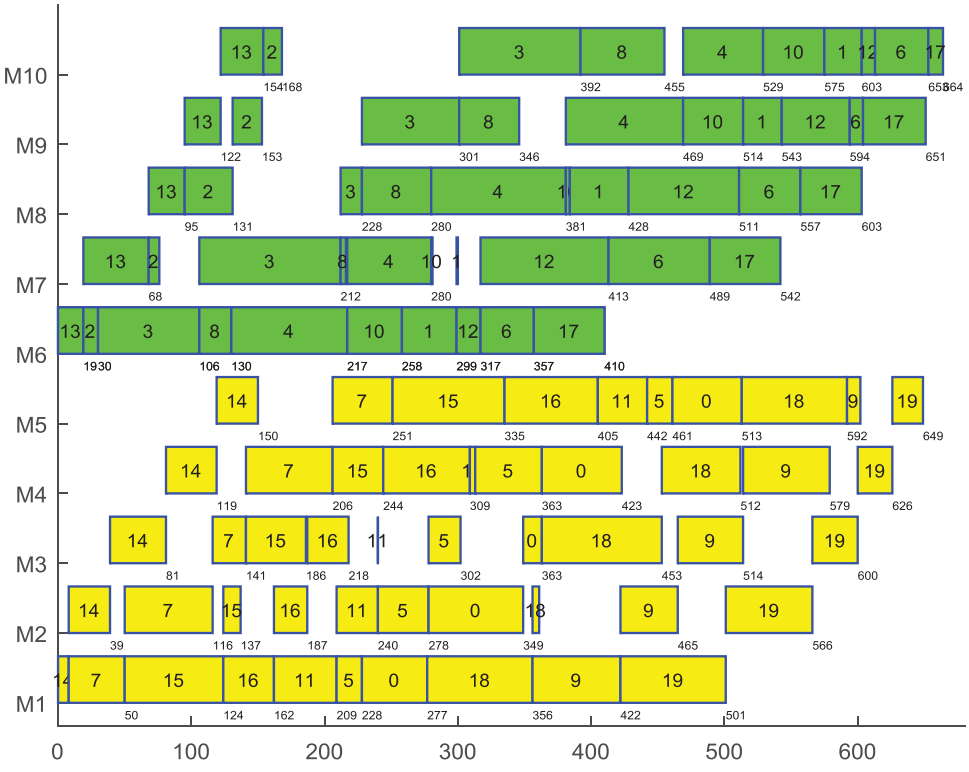


Figure 7. Gantt chart of the best solution for one of the 20–5 instance.

s as a termination criterion, where CPU is set to 10 and 1, respectively, to verify the performance of the proposed algorithm with different CPU time limits. This criterion is practical to apply in realistic production systems.

Tables 7 and 8 provide the comparative results collected from the four algorithms compared with CPU values set to 1 and 10, respectively. In the two tables the first column presents the problem scale with the number of jobs and the number of machines for each group of instances. The next four columns report the RPI values collected from the four algorithms, i.e. VND, BSA, CMA, and IABC. Then, the last four columns give the average time consumed by the four algorithms.

It can be observed from Tables 7 and 8 that (1) in cases involving shorter time limits, the proposed IABC algorithm obtained all optimal values of the given twelve groups of instances; (2) the average performance illustrated by the last line of Table 7 shows that IABC is clearly better than the other three algorithms; (3) in cases involving longer time limits,

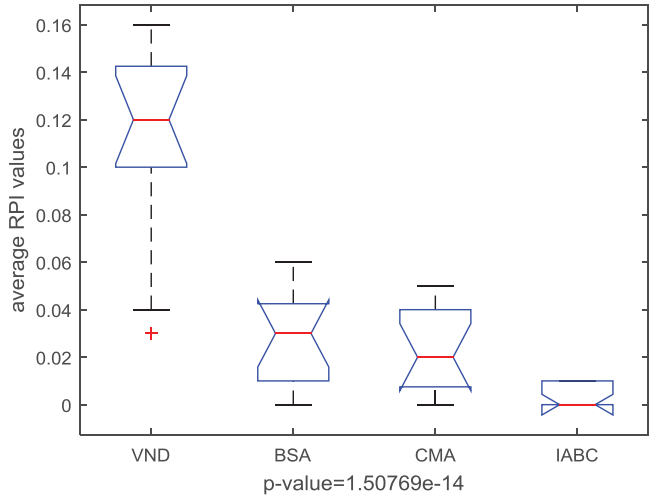


Figure 8. Means and 95% LSD interval comparisons of the four algorithms.

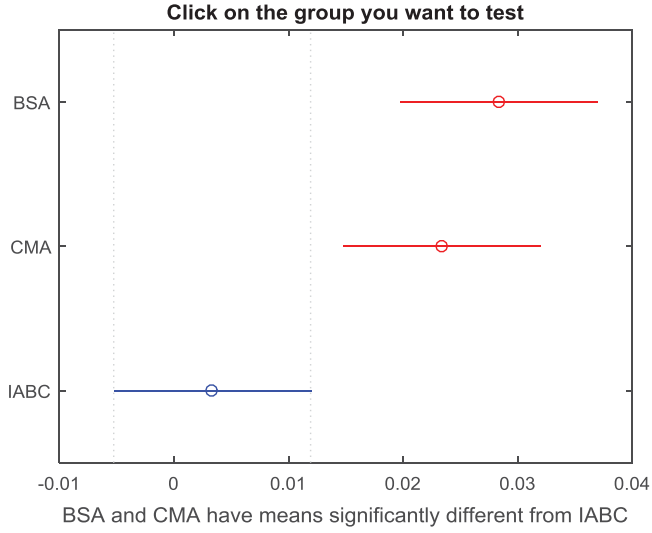


Figure 9. Multi-compare results for the three compared algorithms.

the proposed algorithm also obtained all optimal values, whereas the other three algorithms compared only obtained near optimal solutions. (4) This last line of Table 8 shows the performance of IABC.

Figure 7 presents the Gantt chart of the best solution obtained by the proposed algorithm for the one of the 20–5 instance. Figures 8 and 9 illustrate the means and the 95% LSD interval for RPI values of the four compared algorithms. We can see that the *p*-value is close to zero, revealing significant differences between the compared algorithms.

4.4. Experimental analysis

From the above experimental comparisons, it can be obviously seen that the proposed IABC algorithm shows competitive performance compared with other recently published algorithms. The main advantages of the proposed algorithm are as follows: (1) in the proposed IABC method, the fast DIG method used to compute the fitness of the obtained neighbouring solution shows effective performance, while the other algorithms try to compute the fitness of all possible insertion positions; (2) IABC uses two-dimensional vectors and can permute a solution by simultaneously considering factory assignment and operation scheduling, and therefore, the local search process can be directed to more promising search areas; (3) with well-designed local search operators, IABC can permute and generate a different solution while balancing the factory workload of the system; and (4) to improve the solution which fallen into local best, IABC replaces it with an individual with more promising information and therefore improves the exploration abilities.

5. Conclusions

In this study, an efficient algorithm based on the canonical ABC is proposed as a means of solving distributed flowshop scheduling problems in a prefabricated system. The main contributions of the proposed algorithm are as follows:

(1) Each solution is represented by a two-dimensional vector, where the first dimensional vector is the factory and the second dimensional vector lists the operation scheduling sequence of each factory.
(2) Based on the canonical IG method and considering the distributed problem feature, a distributed IG heuristic is developed.
(3) An efficient population initialisation method that considers the factory workload balance is presented.
(4) A local search approach that randomly swaps two factories for two randomly selected jobs is developed and reveals an optimal position for the two inserting operations via the DIG method.
(5) For the canonical ABC algorithm, using the DIG approach, the main three parts were improved.

The proposed algorithm is tested on different scale problems involving different problem structures. Several efficient algorithms are compared to IABC. Experimental results show the robustness and efficiency of the proposed algorithm. Future work must use the proposed algorithm to solve multi-objective prefabricate scheduling problems. Future works mainly focuses on following issues: (1) to apply the proposed algorithm to solve other types of distributed prefabricated scheduling problems, such as distributed vehicle routing problems in prefabricated system, and distributed optimisation problems with more realistic constraints; (2) to consider two or more objectives and develop a multi-objective optimisation algorithm for solving more realistic green intelligent building optimisation problems; and (3) to include data-driven optimisation methods (Li, Yi et al. 2018) and to solve large-scale scheduling problems.

Disclosure statement

No potential conflict of interest was reported by the authors.

Funding

This research was partially supported by the National Natural Science Foundation of China [grant numbers 61773192, 61773246, and 61803192]; the Shandong Province Higher Educational Science and Technology Program [grant number J17KZ005]; the Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education [grant number K93-9-2017-02]; the State Key Laboratory of Synthetical Automation for Process Industries [grant number PAL-N201602], and major basic research projects in Shandong [grant number ZR2018ZB0419].

References

Anvari, B., P. Angeloudis, and W. Y. Ochieng. 2016. "A Multi-objective GA-based Optimisation for Holistic Manufacturing, Transportation and Assembly of Precast Construction." *Automation in Construction* 71: 226–241.

Arashpour, M., R. Wakefield, B. Abbasi, E. W. M. Lee, and James Minas. 2016. "Off-site Construction Optimization: Sequencing Multiple Job Classes with Time Constraints." *Automation in Construction* 71: 262–270.

Bai, J., and H. Liu. 2016. "Multi-objective Artificial Bee Algorithm Based on Decomposition by PBI Method." *Applied Intelligence* 45 (4): 976–991.

Bargaoui, H., O. B. Driss, and K. Ghédira. 2017. "A Novel Chemical Reaction Optimization for the Distributed Permutation Flowshop Scheduling Problem with Makespan Criterion." *Computers & Industrial Engineering* 111: 239–250.

Benjaoran, V., and N. Dawood. 2003. "Development of an Artificial Intelligence Planner Framework for Bespoke Precast Concrete Production." Proceeding of the 11th Annual Conference of International Group for Lean Construction, Blackburg, VA, July 22–24.

Benjaoran, V., and N. Dawood. 2006. "Intelligence Approach to Production Planning System for Bespoke Precast Concrete Products." *Automation in Construction* 15 (6): 737–745.

Benjaoran, V., N. Dawood, and B. Hobbs. 2005. "Flowshop Scheduling Model for Bespoke Precast Concrete Production Planning." *Construction Management and Economics* 23 (1): 93–105.

Chan, W. T., and H. Hu. 2002. "Production Scheduling for Precast Plants Using a Flow Shop Sequencing Model." *Journal of Computing in Civil Engineering* 16 (3): 165–174.

Chen, C., D. T. Huy, L. K. Tiong, I. M. Chen, and Y. Cai. 2018. "Optimal Facility Layout Planning for AGV-based Modular Prefabricated Manufacturing System." *Automation in Construction*. doi:10.1016/j.autcon.2018.08.008.

Chen, J. H., L. R. Yang, and H. W. Tai. 2016. "Process Reengineering and Improvement for Building Precast Production." *Automation in Construction* 68: 249–258.

Deng, J., and L. Wang. 2017. "A Competitive Memetic Algorithm for Multi-objective Distributed Permutation Flow Shop Scheduling Problem." *Swarm and Evolutionary Computation* 32: 121–131.

Duan, P., J. Li, Y. Wang, H. Sang, and B. Jia. 2018. "Solving Chiller Loading Optimization Problems Using an Improved Teaching-learning-based Optimization Algorithm." *Optimal Control Applications and Methods* 39 (1): 65–77.

Fernandez-Viagas, V., and J. M. Framinan. 2015. "A Bounded-search Iterated Greedy Algorithm for the Distributed Permutation Flowshop Scheduling Problem." *International Journal of Production Research* 53 (4): 1111–1123.

Hu, C. Y., H. Liu, and P. Zhang. 2016. "Cooperative Co-evolutionary Artificial Bee Colony Algorithm Based on Hierarchical Communication Model." *Chinese Journal of Electronics* 25: 570–576.

Joardar, S., A. Chatterjee, S. Bandyopadhyay, and U. Maulik. 2017. "Multi-size Patch Based Collaborative Representation for Palm Dorsa Vein Pattern Recognition by Enhanced Ensemble Learning with Modified Interactive Artificial Bee Colony Algorithm." *Engineering Applications of Artificial Intelligence* 60: 151–163.

Karaboga, D. 2005. "An Idea Based on Honey Bee Swarm for Numerical Optimization[R]." Technical Report-tr06. Erciyes University, Engineering Faculty, Computer Engineering Department.

Karaboga, D., and B. Basturk. 2007. "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm." *Journal of Global Optimization* 39 (3): 459–471.

Ko, C. H., and S. F. Wang. 2011. "Precast Production Scheduling Using Multi-objective Genetic Algorithms." *Expert Systems with Applications* 38 (7): 8293–8302.

Komaki, M., and B. Malakooti. 2017. "General Variable Neighborhood Search Algorithm to Minimize Makespan of the Distributed No-wait Flow Shop Scheduling Problem." *Production Engineering* 11: 315–329.

Kong, L., H. Li, H. Luo, L. Ding, X. Luo, and M. Skitmore. 2017. "Optimal Single-machine Batch Scheduling for the Manufacture, Transportation and JIT Assembly of Precast Construction with Changeover Costs Within Due Dates." *Automation in Construction* 81: 34–43.

Leu, S., and S. Hwang. 2001. "Optimal Repetitive Scheduling Model with Sharable Resource Constraint." *Journal of Construction Engineering and Management* 127 (4): 270–280.

Leu, S. S., and S. T. Hwang. 2002. "GA-based Resource-constrained Flow-shop Scheduling Model for Mixed Precast Production." *Automation in Construction* 11 (4): 439–452.

Li, J. Q., P. Y. Duan, H. Y. Sang, S. Wang, Z. M. Liu, and P. Duan. 2018. "An Efficient Optimization Algorithm for Resource-constrained Steelmaking Scheduling Problems." *IEEE Access* 6: 33883–33894.

Li, J. Q., and Q. K. Pan. 2015. "Solving the Large-scale Hybrid Flow Shop Scheduling Problem with Limited Buffers by a Hybrid Artificial Bee Colony Algorithm." *Information Sciences* 316: 487–502.

Li, J. Q., Q. K. Pan, and P. Y. Duan. 2016. "An Improved Artificial Bee Colony Algorithm for Solving Hybrid Flexible Flowshop with Dynamic Operation Skipping." *IEEE Transactions on Cybernetics* 46 (6): 1311–1324.

Li, J., Q. Pan, and K. Gao. 2011. "Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems." *International Journal of Advanced Manufacturing Technology* 55: 1159–1169.

Li, J. Q., Q. K. Pan, and M. F. Tasgetiren. 2014. "A Discrete Artificial Bee Colony Algorithm for the Multi-objective Flexible Job-shop Scheduling Problem with Maintenance Activities." *Applied Mathematical Modelling* 38 (3): 1111–1132.

Li, J. Q., H. Y. Sang, Y. Y. Han, C. G. Wang, and K. Z. Gao. 2018. "Efficient Multi-objective Optimization Algorithm for Hybrid Flow Shop Scheduling Problems with Setup Energy Consumptions." *Journal of Cleaner Production* 181: 584–598.

Li, J. Q., J. D. Wang, Q. K. Pan, P. Y. Duan, H. Y. Sang, K. Z. Gao, and Y. Xue. 2017. "A Hybrid Artificial Bee Colony for Optimizing a Reverse Logistics Network System." *Soft Computing* 21 (20): 6001–6018.

Li, C. D., J. Q. Yi, H. K. Wang, G. Q. Zhang, and J. Q. Li. 2018. "Interval Data Driven Construction of Shadowed Sets with Application to Linguistic Word Modeling." *Information Sciences*. doi:10.1016/j.ins.2018.11.018.

Lin, J., Z. J. Wang, and X. Li. 2017. "A Backtracking Search Hyper-heuristic for the Distributed Assembly Flow-shop Scheduling Problem." *Swarm and Evolutionary Computation* 36 (2017): 124–135.

Lin, S. W., and K. C. Ying. 2016. "Minimizing Makespan for Solving the Distributed No-wait Flowshop Scheduling Problem." *Computers & Industrial Engineering* 99: 202–209.

Lin, S. W., K. C. Ying, and C. Y. Huang. 2013. "Minimising Makespan in Distributed Permutation Flowshops Using a Modified Iterated Greedy Algorithm." *International Journal of Production Research* 51 (16): 5029–5038.

Liu, H., B. Xu, D. Lu, and G. Zhang. 2018. "A Path Planning Approach for Crowd Evacuation in Buildings Based on Improved Artificial Bee Colony Algorithm." *Applied Soft Computing* 68: 360–376.

Mohammadi, F. G., and M. S. Abadeh. 2014. "Image Steganalysis Using a Bee Colony Based Feature Selection Algorithm." *Engineering Applications of Artificial Intelligence* 31: 35–43.

Montgomery, D. C. 2005. *Design and Analysis of Experiments*. Arizona: John Wiley & Sons.

Naderi, B., and R. Ruiz. 2010. "The Distributed Permutation Flowshop Scheduling Problem." *Computers & Operations Research* 37 (4): 754–768.

Naderi, B., and R. Ruiz. 2014. "A Scatter Search Algorithm for the Distributed Permutation Flowshop Scheduling Problem." *European Journal of Operational Research* 239 (2): 323–334.

Pan, Q. K., M. F. Tasgetiren, P. N. Suganthan, and T. J. Chua. 2011. "A Discrete Artificial Bee Colony Algorithm for the Lot-streaming Flow Shop Scheduling Problem." *Information Sciences* 181 (12): 2455–2468.

Rifai, A. P., H. T. Nguyen, and S. Z. M. Dawal. 2016. "Multi-objective Adaptive Large Neighborhood Search for Distributed Reentrant Permutation Flow Shop Scheduling." *Applied Soft Computing* 40: 42–57.

Sonmez, M., A. P. Akgüngör, and S. Bektaş. 2017. "Estimating Transportation Energy Demand in Turkey Using the Artificial Bee Colony Algorithm." *Energy* 122: 301–310.

Sundar, S., P. N. Suganthan, C. T. Jin, C. T. Xiang, and C. C. Soon. 2017. "A Hybrid Artificial Bee Colony Algorithm for the Job-shop Scheduling Problem with No-wait Constraint." *Soft Computing* 21 (5): 1193–1202.

Tasgetiren, M. F., Q. K. Pan, P. N. Suganthan, and A. Oner. 2013. "A Discrete Artificial Bee Colony Algorithm for the No-idle Permutation Flowshop Scheduling Problem with the Total Tardiness Criterion." *Applied Mathematical Modelling* 37 (10): 6758–6779.

Wang, J. L., B. Gong, H. Liu, and S. H. Li. 2015. "Multidisciplinary Approaches to Artificial Swarm Intelligence for Heterogeneous Computing and Cloud Scheduling." *Applied Intelligence* 43: 662–675.

Wang, K., Y. Huang, and H. Qin. 2016. "A Fuzzy Logic-based Hybrid Estimation of Distribution Algorithm for Distributed Permutation Flowshop Scheduling Problems Under Machine Breakdown." *Journal of the Operational Research Society* 67 (1): 68–82.

Wang, S., L. Wang, M. Liu, and Y. Xu. 2013. "An Effective Estimation of Distribution Algorithm for Solving the Distributed Permutation Flow-shop Scheduling Problem." *International Journal of Production Economics* 145 (1): 387–396.

Wang, L., G. Zhou, Y. Xu, and M. Liu. 2012. "An Enhanced Pareto-based Artificial Bee Colony Algorithm for the Multi-objective Flexible Job-shop Scheduling." *The International Journal of Advanced Manufacturing Technology* 60 (9–12): 1111–1123.

Wang, L., G. Zhou, Y. Xu, and M. Liu. 2013. "A Hybrid Artificial Bee Colony Algorithm for the Fuzzy Flexible Job-shop Scheduling Problem." *International Journal of Production Research* 51 (12): 3593–3608.

Wang, L., G. Zhou, Y. Xu, S. Y. Wang, and M. Liu. 2012. "An Effective Artificial Bee Colony Algorithm for the Flexible Job-shop Scheduling Problem." *The International Journal of Advanced Manufacturing Technology* 60 (1–4): 303–315.

Xu, Y., L. Wang, S. Wang, and M. Liu. 2014. "An Effective Hybrid Immune Algorithm for Solving the Distributed Permutation Flow-shop Scheduling Problem." *Engineering Optimization* 46 (9): 1269–1283.

Yang, Z., Z. Ma, and S. Wu. 2016. "Optimized Flowshop Scheduling of Multiple Production Lines for Precast Production." *Automation in Construction* 72: 321–329.

Ying, K. C., S. W. Lin, C. Y. Cheng, and C. D. He. 2017. "Iterated Reference Greedy Algorithm for Solving Distributed No-idle Permutation Flowshop Scheduling Problems." *Computers & Industrial Engineering* 110: 413–423.

Zeng, Y., Z. Zhang, and A. Kusiak. 2015. "Predictive Modeling and Optimization of a Multi-zone HVAC System with Data Mining and Firefly Algorithms." *Energy* 86: 393–402.

Zhang, R., P. C. Chang, S. Song, and C. Wu. 2017. "A Multi-objective Artificial Bee Colony Algorithm for Parallel Batch-processing Machine Scheduling in Fabric Dyeing Processes." *Knowledge-Based Systems* 116: 114–129.

Zhang, P., H. Liu, and Y. Ding. 2014. "Dynamic Bee Colony Algorithm Based on Multi-species Co-evolution." *Applied Intelligence* 40: 427–440.

Zhang, G., K. Xing, and F. Cao. 2017. "Scheduling Distributed Flowshops with Flexible Assembly and Set-up Time to Minimise Makespan." *International Journal of Production Research*, 1–19. doi:10.1080/00207543.2017.1401241.

Zheng, Z., and J. Li. 2018. "Optimal Chiller Loading by Improved Invasive Weed Optimization Algorithm for Reducing Energy Consumption." *Energy and Buildings* 161: 80–88.

Zheng, Z., J. Li, and P. Duan. 2019. "Optimal Chiller Loading by Improved Artificial Fish Swarm Algorithm for Energy Saving." *Mathematics and Computers in Simulation* 155: 227–243.

Zhou, J., and X. Yao. 2017a. "Multi-objective Hybrid Artificial Bee Colony Algorithm Enhanced with Lévy Flight and Self-adaption for Cloud Manufacturing Service Composition." *Applied Intelligence* 47: 721–742.

Zhou, J., and X. Yao. 2017b. "Multi-population Parallel Self-adaptive Differential Artificial Bee Colony Algorithm with Application in Large-scale Service Composition for Cloud Manufacturing." *Applied Soft Computing* 56: 379–397.